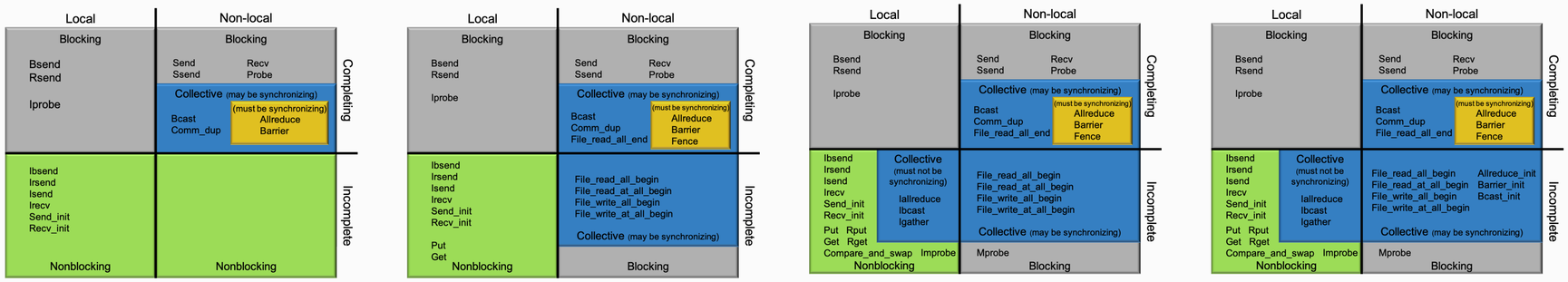
MPI Semantic Terms and Conventions Explained

MPI Semantic Terms and Conventions Explained

Claudia Blaas-Schenner[1], Daniel J. Holmes[2], Rolf Rabenseifner[3], Anthony Skjellum[4], Guillaume Mercier[5], Julien Jaeger[6], Purushotham V. Bangalore[7]
VSC/TU Wien[1], EPCC[2], HLRS/U. Stuttgart[3], UTC[4], BIT/INRIA[5], CEA/DAM/DIF[6], UAB[7]

The Big Idea: Understanding Semantic Terms and Conventions is Key to Using, Extending, and Implementing MPI Correctly

The Evolution of MPI Semantics



Procedures

An **MPI procedure is incomplete** if it may return before the associated operation has finished its completion stage, which implies that the user is not allowed to reuse parameters (such as buffers) specified when initializing the operation. Therefore, an incomplete procedure only includes the initialization and/or starting stages.

An **MPI procedure is completing** if return from the procedure indicates that the associated operation has finished its completion stage, which implies that the user can rely on the content of the output message data buffers and modify the content of input and output message data buffers.

If a completing procedure is not also a freeing procedure then the user is not permitted to deallocate the message data buffers or to modify the array arguments. Procedures not associated with an operation are also defined to be completing.

An **MPI procedure is freeing** if return from the procedure indicates that the associated operation has finished its freeing stage, which implies that the user can reuse all parameters specified when initializing the associated operation.

An **MPI procedure is local** if it returns control to the calling MPI process based only on the state of the local MPI process that invoked it. Local procedures may be of short or long duration, but their behavior is wholly independent of the activity of other MPI processes or procedure invocations.

An **MPI procedure is non-local** if returning may require the execution of some MPI procedure on another MPI process. Such procedures may require communication occurring with another MPI process.

An **MPI procedure is blocking** if it is completing, and/or freeing, and/or non-local.

An **MPI procedure is nonblocking** if it is incomplete and local.

Operation Stages

Initialization: The initialization stage hands over the argument list to the operation but not message data buffer(s) contents. An operation's specification may state that array arguments must remain unchanged till the operation is freed.

Starting: The starting stage hands over the control of the message data buffer to the associated operation.

Note that the term **initiating** in MPI refers to the combination in sequence of the initialization and starting stages.

Completion: The completion stage returns control of the content of the message data buffer(s) to the application and indicates that any output buffers have been updated.

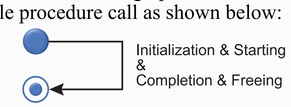
Note that an MPI operation is **complete** when the MPI procedure implementing the completion stage returns.

Freeing: The freeing stage returns control of the rest of the argument list (e.g., the buffer address and array arguments).

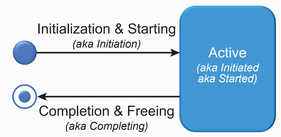
MPI Operation:	Blocking	Nonblocking	Persistent
1. Initialization	MPI_SEND *(Non-local)* or MPI_SSEND *(Non-local)* or MPI_BSEND *(Local)* or MPI_RSEND *(Local)* or MPI_RECV *(Non-local)* or MPI_BCAST *(Non-local)* *(Completing + Freeing)*	MPI_ISEND *(Local)* or MPI_IRECV *(Local)* or MPI_IBCAST *(Local)* *(Incomplete)*	MPI_SEND_INIT *(Local)* or MPI_RECV_INIT *(Local)* or MPI_BCAST_INIT *(Non-local)* *(Incomplete)*
2. Starting			MPI_START *(Local)* *(Incomplete)*
3. Completion		MPI_WAIT *(Non-local)* *(Completing + Freeing)*	MPI_WAIT *(Non-local)* *(Completing)*
4. Freeing			MPI_REQUEST_FREE *(Local)* *(Freeing)*

Types of Operation

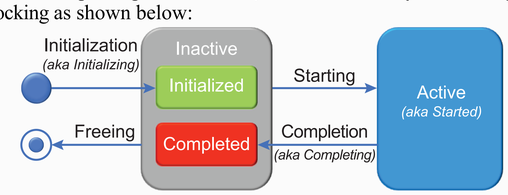
Blocking Operation For a blocking operation, all four stages are combined in a single procedure call as shown below:



Nonblocking Operation For a nonblocking operation, the initialization and starting stages are combined into a single nonblocking procedure call and the completion and freeing stages are combined into a separate, single procedure call, which can be blocking or nonblocking as shown below:



Persistent Operation For a persistent operation, all four stages are effected with separate procedure calls, each of which may be blocking or nonblocking as shown below:



Why this all matters in practice

- Users can be confused how MPI operations should behave and how to use them.
- Implementors can, in principle, make suboptimal decisions about the meaning of MPI operations with regard to progress, resources, …
- What's really going on with resources in MPI operations, and scope of resource ownership is important and must be clear (e.g., vectors of indices).
- Holes in the MPI standard can/will be discovered and be addressed.
- Proposed MPI operations can be categorized according to the semantic terms and conventions… and be graded for compliance with these core concepts.
- Opportunities to improve, orthogonalize, and extend MPI are key to its future value in Exascale and beyond.

Key Observations

Surprisingly, important concepts remain insufficiently specified, ambiguous, and in some cases, inconsistent or conflicting despite 26 years of standardization, implementation, and utilization.

This poster explains these concepts, and mentions some areas still to be worked on, emphasizing understandability. It breaks down, complements and supplements our EuroMPI 2019 paper on the same subject matter.

Up to MPI-3.1, a blocking MPI procedure might not actually block in the traditional sense.

The MPI definitions of the term blocking only discuss usage of parameters given to procedures, they do not include any mention of delaying the return of the procedure until a matching MPI procedure is called at another MPI process. A blocking procedure is permitted to return as soon as the supplied parameters can be reused.

Up to MPI-3.1, a nonblocking MPI procedure might actually block in the traditional sense.

The term nonblocking is not formally a word in English, but its meaning can be extrapolated as the opposite of blocking; that is, it would seem reasonable to assume that nonblocking means will not block or must not block. However, as with the term blocking, these MPI definitions of the term nonblocking only speak to the usage of parameters given to procedures; they do not include any mention of delaying the return of the procedure until a matching completing MPI procedure is called at the same MPI process.

There is no such thing as a persistent MPI procedure; only MPI operations can be persistent.

The concept of locality classifying an MPI procedure as **local** or **non-local** has always meant what the English words suggested.

Future Work: Further Semantic Terms and Conventions to Be Studied and Refined

- Why Persistent initialization calls are non-local (and implications of changing to local)?
- Why nonblocking constructors and destructors ~ "orthogonality" ~ are needed/useful standard-wide?
- Defining/Refining/Clarifying MPI Process, Rank, and Thread: What's valid, possible, needed?
- The MPI Standard states: two send (or receive) operations on two different threads are "logically concurrent even if one physically precedes" the other. What does that mean in practice?